Parcurgeri Euler. Labirinturi

Abstract

În acest articol se analizează probleme legate de procesul ieşirii dintr-un labirint, extinzându-se algoritmi cunoscuți de parcurgere în sensul deplasării 'din cameră în cameră'. Se extind parcurgerea în adâncime şi parcurgerea D, se prezintă un algoritm de determinare a bazei de cicluri fundamentale proiectat după acelaşi principiu, prezentându-se legătura cu problema comis voiajorului chinez.

Introducere

Definiție 1: Considerăm un labirint ca fiind o construcție pătratică formată din camere (pătrățele). Un labirint se poate codifica printr-o matrice A de ordin N în care intrarea A[I, J] este un număr între 0 şi 15 cu următoarea semnificație: Dacă se consideră vecinii camerei [I, J] în ordinea nord, est, sud, vest şi se reține 1 în cazul existenței uşii către vecinul respectiv şi 0 în caz contrar, numărul de patru cifre în baza doi astfel format şi trecut în baza 10 este tocmai A[I, J].

Observație: O uşa se poate deschide în ambele direcții, deci nu orice tablou N*N cu numere între 0 şi 15 codifică un labirint; numerele camerelor nu sunt total independente.

Definiție 2: Considerăm un labirint ca fiind o matrice pătratică A de ordin N cu elemente 0 şi 1 cu semnificația; 0 reprezintă spațiu liber iar 1 reprezintă zid.

Prima definiție este mai naturală, dar este mai greu de lucrat cu ea, a doua definiție prezintă avantajul unei facile generări aleatoare de labirinturi. Ambele definiții corespund unor labirinturi planare. Aceste definiții se pot generaliza prin termeni de teoria grafurilor:

Definiție 3: Se numeşte labirint un graf neorientat G=(V,M) împreună cu o submulțime de noduri *distinse* numite noduri de plecare.

În continuare vom presupune că graful este conex şi există un unic nod de plecare.

Algoritmi în cazul labirinturilor planare

Considerăm labirintul conform primelor două definiții. În cadrul algoritmilor nu se foloseşte matricea A, poziția curentă în labirint fiind exprimată printr-o singură valoare I, făcându-se abstracție de coordonatele sale. Definim patru direcții N, E, S, V şi o funcție succesor în modul următor: succ('N')='E', succ('E')='S', succ('S')='V', succ('V')='N'.

Pentru a simula parcurgerea în adâncime nu este necesară menținerea unei stive. În locul acesteia folosim un vector de marcaj: `SEL`. Vectorul `SEL` reține prin valori true un drum între poziția curentă şi poziția de plecare. Se mai foloseşte vectorul `SEL_VIZ` cu elemente logice. În timpul executării algoritmului pozițiile vizitate vor fi marcate prin vectorul `SEL_VIZ`. Algoritmul este următorul:

```
algoritm df( I: pozitie );

{ vizit ( I );
   SEL_VIZ[I}:=true;
   write( I );
   PRED:=0;
   SEL[I]:=true;
```

```
   D:='N';
   repeat
         D←succ(D);
         while  [nu exista nod J pe directia D] or
                    [exista J, SEL[J]=true si (J<>PRED)] do
                 D←succ(D);
          J:=acel nod de pe directia D;
          if  (SEL_VIZ[I]=false) then
                  { vizit( I ); SEL_VIZ[I]:=true; }
          if  (J=PRED)  then SEL[I]:=false;
                                 else  SEL[I]:=true;
          I:=J;
          write ( I );
          if [exista P vecin al lui I cu SEL[P]=true] then
               PRED:=P;
          else stop;
    until (false);
}
```

Pentru fiecare nod, după vizitare, îi calculăm predecesorul ca fiind acel unic nod vecin care este şi marcat. Singurul nod care nu are predecesor în acest punct va fi nodul de pornire (este totuşi şi el marcat). După calculul predecesorului, calculăm succesorul nodului I prin tehnica prima la stânga. Sunt două cazuri, după cum acest succesor coincide sau nu cu predecesorul, cazuri tratate corespunzător actualizării "firului Ariadnei" format din nodurile selectate la un moment dat.

Notăm cu N numărul de camere accesibile din poziția de plecare. Numim parcurgere Euler un circuit care trece prin toate camerele, nu neapărat o singură dată şi pentru care orice două poziții succesive reprezintă camere alăturate. În raport cu operația fundamentală de trecere dintr-o cameră în alta, algoritmul are ordinul de complexitate O(N), producând la ieşire un circuit cu 2*(N-1) muchii, circuit ce parcurge arborele DF. Circuitul parcurgerii Euler este listat prin apeluri `write`. Ca memorie suplimentară se folosesc doar doi vectori de biți.

Prezentăm al doilea algoritm în care presupunem că labirintul formează un arbore. Atunci pentru parcurgerea sa nu mai este necesară folosirea vectorului `SEL` de marcaj. Algoritmul în pseudocod este următorul:

```
algoritm df_arbore( Io: pozitie );

{ I:=Io;
   write( I );
   D:='N';
    repeat
           D:=succ(D);
           while [nu exista nod J pe directia D] do
                      D:=succ(D);
           J:=acel nod de pe directia D;
           I:=J;
           write( I );
     until (I=Io);
}
```

În cazul existenței unui posibil ciclu în labirint acest algoritm poate intra într-un ciclu infinit. Aşadar acesta este un algoritm euristic. Există camere vizitate de mai multe ori. O vizitare este indicată de un apel `write`. Ordinul de complexitate este O(N), unde N reprezintă numărul de noduri ale arborelui. Cu ajutorul unui vector `SEL_VIZ` putem produce vizitarea camerelor labirintului o singura dată.

Cazul general. O interpretare intuitivă

Un labirint este format din camere şi coridoare de trecere (uşi). Presupunem că într-o anumită cameră se află o persoană. În acest moment persoana nu are acces decât la informațiile prezente în camera respectivă. Printre aceste informații sunt şi acelea legate de existența uşilor către camerele vecine. Persoana poate prelucra informațiile pe un calculator, odată cu deplasarea sa prin labirint, în scopul culegerii de date. Vom privi aşadar un labirint ca fiind un graf cu informație incompletă. Camerele vor fi nodurile grafului iar uşile muchiile sale.

Definiție Labirint. Se numeşte labirint un graf neorientat G=(V,M) împreună cu o submulțime de noduri *distinse* numite noduri de plecare. Presupunem că graful este conex şi există un unic nod de plecare.

Cunoaştem că, în cazul grafurilor, o parcurgere înseamnă o listare (sau alt tip de prelucrare) a tuturor nodurilor, fiecare nod fiind vizitat o singură dată. În cazul unui labirint putem introduce în mod natural noțiunea de parcurgere Euler.

Definiție Parcurgere Euler. Se numeşte parcurgere Euler un circuit care conține toate nodurile grafului împreună cu o alegere pe acest circuit a vizitării fiecărui nod o singură dată.

Prin această definiție se extinde noțiunea de parcurgere. Cu alte cuvinte deplasarea în scopul parcurgerii se va efectua 'din cameră în cameră'. Dacă avem dată o anumită parcurgere, o parcurgere Euler ce vizitează în aceeaşi ordine nodurile grafului se va numi parcurgere Euler conformă cu respectiva parcurgere. Dacă avem pentru început doar circuitul parcurgerii Euler şi labirintul are un singur nod de plecare, putem forma o primă parcurgere Euler prin vizitarea fiecărui nod prima dată când intrăm în el.

Parcurgeri Euler de vârfuri

În continuare vom referi parcurgerea Euler definită anterior ca fiind o parcurgere Euler de vârfuri. Ne punem problema dacă în cazul unor parcurgeri clasice putem găsi algoritmi care generează o parcurgere Euler conformă cu respectiva parcurgere.

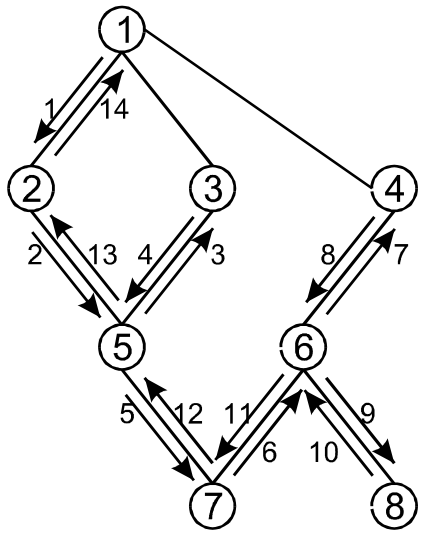
Cazul parcurgerii în adâncime (DF).Parcurgerea DL (Deapth Last)

S-a prezentat anterior algoritmul parcurgerii Euler conforme parcurgerii DF în cazul particular al unui labirint pe patru direcții în plan. Să dăm acum algoritmul pe cazul general al unui graf:

```
algoritm DF_EULER ( I );

{ vizit [ I ]; SEL [ I ]←true;
   for [ toţi J vecini ai lui I ) do
           if  (SEL [ J ]=false) then
                 { DF_EULER ( J );
                    vizit [ I ];
                                          }
}
```

Pe exemplu de graf din figura următoare , circuitul parcurgerii Euler (produs de vizitări) este: 1, 2, 5, 3, 5, 7, 6, 4, 6, 8, 6, 7, 5, 2, 1.



Putem însă să ne imaginăm că vizitarea se produce la ultima trecere prin fiecare cameră. Ca exemplu intuitiv: persoana doreşte să mătură toate camerele şi să strângă gunoiul în camera de plecare. Plecând de la acelaşi circuit, se produce o altă parcurgere de graf .O vom numi parcurgerea DL(Deapth Last). Iată procedura ce o realizează:

```
algoritm DL ( I );

{ SEL [ I ] ←1;
   for [ toţi J vecini ai lui  I ] do
           if (SEL[ J ]=false) then
                 DL ( J );
   vizit [ I ];
}
```

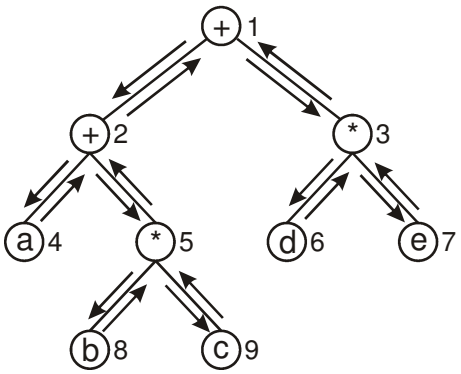
Sintetizând rezultatele pe graful prezentat obţinem:

Parcurgerea DL : 3, 4, 8, 6, 7, 5, 2, 1

circuitul parcurgerii Euler : $\underline{1}\ \underline{2}\ 5\ \underline{\overline{3}}\ 5\ \underline{7}\ \underline{6}\ \underline{\overline{4}}\ 6\ \underline{\overline{8}}\ \overline{6}\ \overline{7}\ \overline{5}\ \overline{2}\ \overline{1}$

Parcurgerea DF : 1, 2, 5, 3, 7, 6, 4, 8

Observăm că parcurgerea DL (Deapth Last) generalizează conceptul de parcurgere în postordine a unui arbore. Pentru a sublinia aceste lucru, prezentăm pe o figură cazul arborelui asociat unei expresii aritmetice şi forma poloneză postfixată respectivă ca fiind rezultatul unei parcurgeri DL.



Parcurgerea DL , SDR:

4, 8, 9, 5, 2, 6, 7, 3, 1
a, b, c, *, +, d, e, *, +

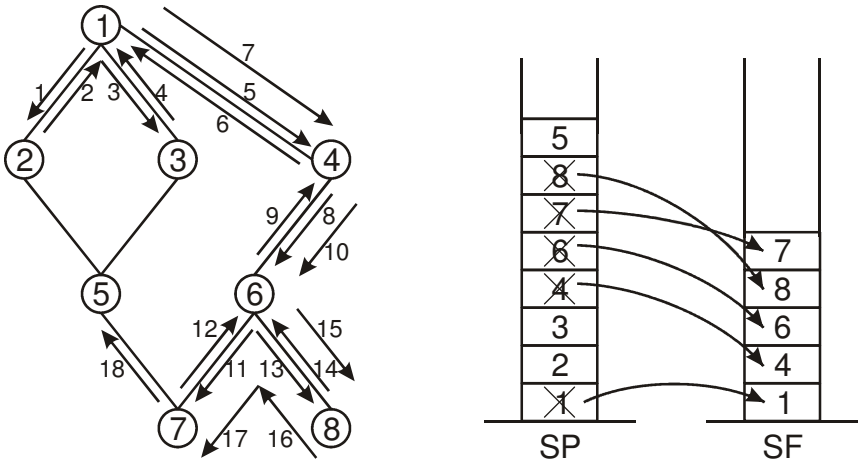
Cazul parcurgerii D (Deapth). Parcurgerea DD (Double deapth)

Vom folosi două stive de lucru: stiva primă (SP) şi stiva finală (SF). Nodurile grafului vor trece succesiv prin cele două stive iar algorimul se încheie prin vidarea stivei finale.

```
algoritm D_Euler ( I );

{ SF ←vidă; SP←vidă;
   SP← I;
   SEL [ I ]:=true;
   repetă
      I←SP;
      vizit [ I ];  /* linia 1
      SF← I;
      for [ toţi J vecini ai lui I ) do
           if (SEL [ J ]=false) then
                { vizit [ J ];  /* linia 2
                   SP← J;
                   SEL [ J ]:=true;
                   vizit [ I ];
                                          }
      while ( TOP(SP) neadiacent cu TOP( SF) ) do
                { I←SF;
                   vizit [ TOP(SF) ];
                                          }
     until ( SF = vidă );
}
```

Prezentăm pe exemplul anterior de graf o porţiune iniţială a circuitului parcurgerii Euler împreună cu modificările realizate până în acel moment în cele două stive:



Dacă vizitarea nodurilor se produce doar la intrarea în prima stivă (linia 1) se regăseşte parcurgerea D. Dacă vizităm nodurile numai la introducerea în stiva finală (linia 2) se produce o nouă parcurgere pe care am denumit-o Double Deapth.

Sintetizând rezultatele pe graful din exemplu obţinem:

Parcurgerea DD : 1, 4, 6, 8, 7, 5, 3, 2

circuitul parcurgerii Euler $\overline{1}\ \underline{2}\ 1\ \underline{3}\ 1\ \underline{4}\ 1\ \overline{4}\ \underline{6}\ 4\ \overline{6}\ \underline{7}\ 6\ \underline{8}\ 6\ \overline{8}\ 6\ \underline{7}\ 5\ \overline{7}\ \overline{5}\ 7\ 6\ 4\ 1\ \overline{3}\ 1\ \overline{2}\ 1$

Parcurgerea D : 1, 2, 3, 4, 6, 7, 8, 5

Încheiem secţiunea cu următoarea remarcă: în foarte multe cazuri executarea unei funcţii recursive revine la o parcurgere Euler de vârfuri pe acest arborele asociat. În cazul în care arborele asociat funcţiei recursive este un arbore binar, această parcurgere Euler este conformă cu parcurgerile în preordine, inordine şi postordine.

Parcurgeri Euler de muchii

În mod natural introducem noţiunea de parcurgere Euler de muchii:

Definiţie Parcurgere Euler de muchii. Se numeşte parcurgere Euler de muchii un circuit care trece cel puţin o dată prin toate muchiile grafului.

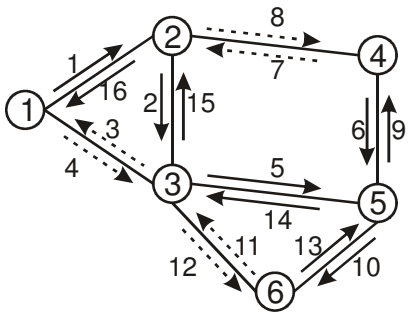
Nu se mai pune imediat problema generalizării unei parcurgeri, lipsind din definiţie alegerea nodurilor ori a muchiilor grafului pe circuit. Semnalăm faptul că această noţiune reprezintă o adevărată liniarizare a unui graf. Cu alte cuvinte reprezintă o stocare a informaţiilor din graf sub formă liniară, informaţii suficiente pentru a reface graful de pornire. Similar cum din forma poloneză postfixată se poate reface arborele asociat unei expresii aritmetice.

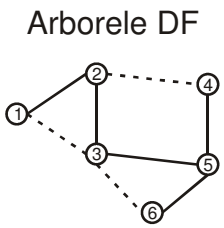
Prezentăm un prim exemplu de parcurgere Euler de muchii obţinut prin modificarea algoritmului de parcurgere în adâncime. O vom numi parcurgere Euler DF de muchii.

```
algoritm muchii_Euler( I )

{ SEL [ I ]:=true;
  vizit [ I ];
  for [ toţi J vecini ai lui I ] do
     if (SEL [ J ]=false) then
                      { muchii_Euler( J ); vizit [ I ]; }
                                        else
                           { vizit [ J ]; vizit [ I ]; }
}
```

Pe graful urmator parcurgerea Euler DF de muchii este următoarea:
1, 2, 3, 1, 3, 5, 4, 2, 4, 5, 6, 3, 6, 5, 3, 2, 1.





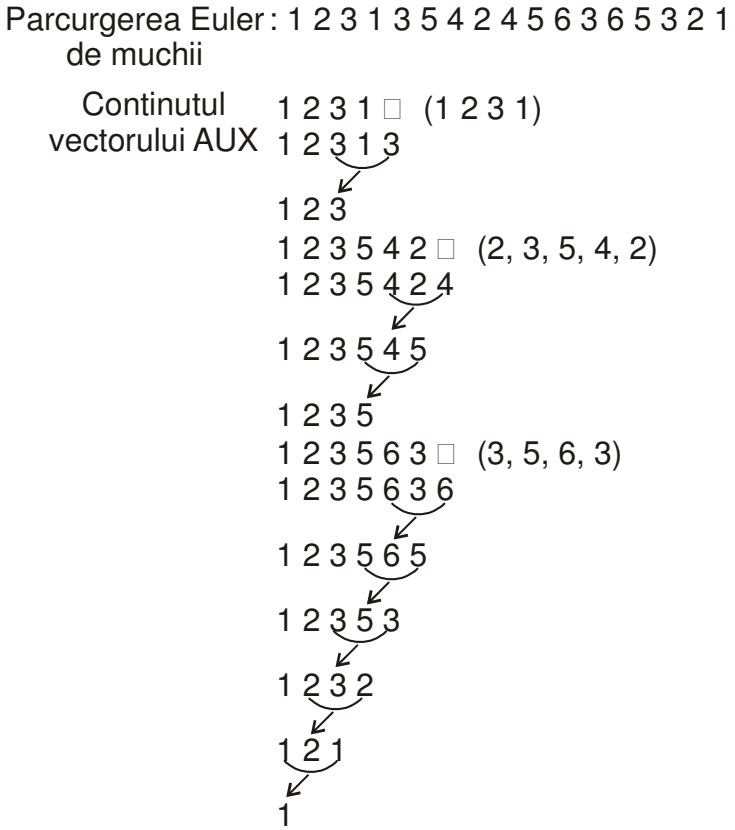
Determinarea unei baze de cicluri fundamentale

Ca prim pas al algoritmului se obţine parcurgerea Euler DF de muchii. Apoi vom prelucra vectorul de noduri obţinut în modul următor :

```
algoritm cicluri ( X: vector; L: lungime );
AUX: vector; K, POZ: întreg; GASIT:logic;

{ K:=0;
   for I:=1 to L do
          {     K:=K+1;
           AUX [ K ]←X [ I ];
     GASIT:=false;
     for J:=1 to K-1 do
          if (AUX [ J ]=AUX [ I ] then
                { GASIT:=true;
                   POZ:=J;
                }
     if  (GASIT) then
          if (POZ=K-2) then
                K:=K-2; // un ciclu are mai mult de două muchii
                          else
                write( 'Ciclul :',AUX[POZ], AUX[POZ+1],..., AUX[
K ])
     } // K va lua valoarea 1
}
```

Pe exemplul anterior de graf, prezentăm conținutul vectorului `aux` în diferite puncte ale execuției algoritmului:



Problema comis-voiajorului chinez

Problemă Fiind dat un graf neorientat cu costuri pe muchii să se obțină un traseu care trece cel puțin o dată prin fiecare muchie şi are costul minim.

Conform noțiunilor prezentate în acest articol problema cere determinarea unei parcurgeri Euler de muchii optimă din punct de vedere al costului. Se cunoaşte rezolvarea polinomială a acestei probleme.

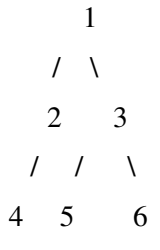
Ne putem întreba dacă o astfel de rezolvare polinomială este posibilă în cazul în care punem condiția ca circuitul să treacă cel puțin o dată prin fiecare nod. Cu alte cuvinte o parcurgere Euler de vârfuri optimă.

Răspuns: Dacă am rezolva polinomial problema pe noduri am produce în particular pe clasa grafurilor hamiltoniene un algoritm polinomial pentru determinarea circuitului hamiltonian de cost minim. În concluzie problema determinării unei parcurgerii Euler optime de vârfuri este NP .

Pentru a studia complexitatea algoritmilor descrişi alegem ca operație fundamentală deplasarea unei persoane dintr-o cameră în alta. În cazul parcurgerii Euler DF realizăm 2*(N-1) deplasări, deci ordinul de complexitate este O(N). În cazul parcurgerii Euler D realizăm 4*(N-1) deplasări deci ordinul de complexitate este tot O(N). Pentru parcurgerea Euler DF a muchiilor avem nevoie de 2*M deplasări, deci ordinul de complexitate este O(M), unde M este numărul de muchii.

Parcurgerea Euler a unui arbore binar

Putem imagina o parcurgere care vizitează toate vârfurile arborelui, la fiecare pas următorul vârf vizitat fiind descendentul sau ascendentul direct al vârfului precedent. Dacă nodul este fără descendenți va apărea o singură dată în parcurgere, dacă are descendenți va apărea şi înaintea nodurilor subarborelui stâng şi după nodurile subarborelui drept. Pentru arborele:



Parcurgerea Euler este : `1, 2, 4, 2, 1, 3, 5, 3, 6, 3, 1.`

Anumite vârfuri vor fi vizitate de mai multe ori. Selectând pe parcurs anumite vârfuri, putem obține subşirurile caracteristice parcurgerii în preordine respectiv postordine. Operația poate fi inversată. Din lista vârfurilor corespunzătoare parcurgerii Euler, putem construi în memorie arborele de la care s-a plecat. Astfel, putem obține în vectorul X şirul vârfurilor în parcurgerea Euler:

```
intreg L=0;

algoritm parcurgere_euler(A:arbore);
{ if (A<>NIL) then
   {L:=L+1; X[L]:=A->INF;
   parcurgere_euler(A->ST);
   if (A->ST<>NIL)and (A->DR<>NIL) then
     {L:=L+1;X[L]:=A->INF;}
   parcurgere_euler(A->DR);
   if (A->ST<>NIL)or(A->DR<>NIL) then
     {L:=L+1;X[L]:=A->INF;}
   }
}
```

Invers, din vectorul X putem obține arborele. Ținem cont că în cazul în care un anumit nod are inițial un singur descendent, în arborele astfel obținut va fi descendentul stâng. Astfel:

```
algoritm creare_euler(P,U:intregi):arbore; M,I:integer;Q:arbore;
{
if (P<=U) then
   { new(Q);Q->INF:=X[P];
     if (P<U) then
       {M:=0;
       for I:=P+1 to U-1 do
            if X[I]=X[P] then M:=I;
         if M<>0 then {
               Q->ST:=creare_euler(P+1,M-1);
               Q->DR:=creare_euler(M+1,U-1);
               }
       else { Q->ST:=creare_euler(P+1,U-1);
```

```
              Q->DR:=NIL;
            }
        }
else {Q->ST:=NIL;
      Q->DR:=NIL;
     }
creare_euler:=Q;
   }
}
```